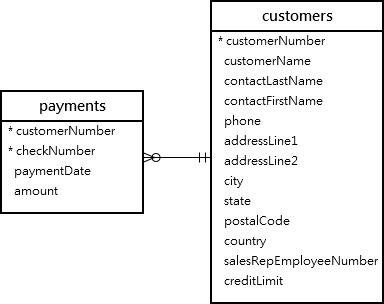
MySQL Views

**Here**, you will learn about MySQL views and how to manipulate views effectively.

## **Introduction to MySQL Views**

Let’s see the following tables customers and payments from the sample database.



This query returns data from both tables customers and payments using the inner join:

SELECT

customerName,

checkNumber,

paymentDate,

amount

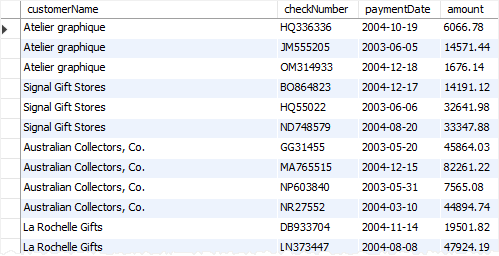
FROM

customers

INNER JOIN

payments USING (customerNumber);

Here is the output:



Next time, if you want to get the same information including customer name, check number, payment date, and amount, you need to issue the same query again.

One way to do this is to save the query in a file, either .txt or .sql file so that later you can open and execute it from MySQL Workbench or any other MySQL client tools.

A better way to do this is to save the query in the database server and assign a name to it.

This named query is called a **database view,** or simply, **view**.

By definition, a view is a named query stored in the database catalog.

To create a new view you use the CREATE VIEW statement. This statement creates a view customerPayments based on the above query above:

CREATE VIEW customerPayments

AS

SELECT

customerName,

checkNumber,

paymentDate,

amount

FROM

customers

INNER JOIN

payments USING (customerNumber);

Once you execute the CREATE VIEW statement, MySQL creates the view and stores it in the database.

Now, you can reference the view as a table in SQL statements. For example, you can query data from the customerPayments view using the SELECT statement:

SELECT \* FROM customerPayments;

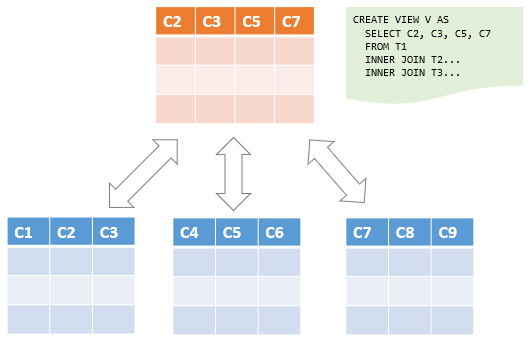
As you can see, the syntax is much simpler.

**Note that a view does not physically store the data**. When you issue the SELECT statement against the view, MySQL executes the underlying query specified in the view’s definition and returns the result set.

For this reason, sometimes, a view is referred to as a virtual table.

MySQL allows you to create a view based on a SELECT statement that retrieves data from one or more tables.

This picture illustrates a view based on columns of multiple tables:



In addition, MySQL even allows you to create a view that does not refer to any table.

But you will rarely find this kind of view in practice.

For example, you can create a view called daysofweek that return 7 days of a week by executing the following query:

CREATE VIEW daysofweek (day) AS

SELECT 'Mon'

UNION

SELECT 'Tue'

UNION

SELECT 'Web'

UNION

SELECT 'Thu'

UNION

SELECT 'Fri'

UNION

SELECT 'Sat'

UNION

SELECT 'Sun';

And you can query data from the daysofweek view as follows:

SELECT \* FROM daysofweek;

This picture shows the output:

https://www.mysqltutorial.org/wp-content/uploads/2019/08/MySQL-View-reference-no-table-example.png

## **Advantages of MySQL Views**

MySQL views bring the following advantages.

### **1) Simplify complex query**

Views help simplify complex queries. If you have any frequently used complex query, you can create a view based on it so that you can reference to the view by using a simple SELECT statement instead of typing the query all over again.

### **2) Make the business logic consistent**

Suppose you have to repeatedly write the same formula in every query.  Or you have a query that has complex business logic. To make this logic consistent across queries, you can use a view to store the calculation and hide the complexity.

### **3) Add extra security layers**

A table may expose a lot of data including sensitive data such as personal and banking information.

By using views and privileges, you can limit which data users can access by exposing only the necessary data to them.

For example, the table employees may contain SSN and address information, which should be accessible by the HR department only.

To expose general information such as first name, last name, and gender to the General Administration (GA) department, you can create a view based on these columns and grant the users of the GA department to the view, not the entire table employees .

### **4) Enable backward compatibility**

In legacy systems, views can enable backward compatibility.

Suppose, you want to normalize a big table into many smaller ones. And you don’t want to impact the current applications that reference the table.

In this case, you can create a view whose name is the same as the table based on the new tables so that all applications can reference the view as if it were a table.

Note that a view and table cannot have the same name so you need to drop the table first before creating a view whose name is the same as the deleted table.

## **Introduction to MySQL CREATE VIEW statement**

The CREATE VIEW statement creates a new view in the database. Here is the basic syntax of the CREATE VIEW statement:

CREATE [OR REPLACE] VIEW [db\_name.]view\_name [(column\_list)]

AS

select-statement;

In this syntax:

First, specify the name of the view that you want to create after the CREATE VIEW keywords. The name of the view is unique in a database. Because views and tables in the same database share the same namespace, the name a view cannot the same as the name of an existing table.

Second, use the OR REPLACE option if you want to replace an existing view if the view already exists. If the view does not exist, the OR REPLACE has no effect.

Third, specify a list of columns for the view. By default, the columns of the view are derived from the select list of the SELECT statement. However, you can explicitly specify the column list for the view by listing them in parentheses following the view name.

Finally,  specify a SELECT statement that defines the view. The SELECT  statement can query data from tables or views. MySQL allows you to use the ORDER BY clause in the SELECT statement but ignores it if you select from the view with a query that has its own ORDER BY clause.

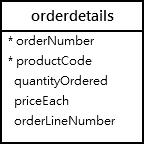
By default, the CREATE VIEW statement creates a view in the current database. If you want to explicitly create a view in a given database, you can qualify the view name with the database name.

## **MySQL CREATE VIEW examples**

Let’s take some example of using the CREATE VIEW statement to create new views.

### **1) Creating a simple view example**

Let’s take a look at the orderDetails table from the sample database:



This statement uses the CREATE VIEW statement to create a view that represents total sales per order.

CREATE VIEW salePerOrder AS

SELECT

orderNumber,

SUM(quantityOrdered \* priceEach) total

FROM

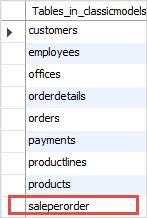
orderDetails

GROUP by orderNumber

ORDER BY total DESC;

If you use the SHOW TABLE command to view all tables in the classicmodels database, you will see the view salesPerOrder is showing up in the list.

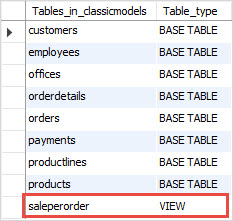
SHOW TABLES;



This is because the views and tables share the same namespace as mentioned earlier.

To know which object is a view or table, you use the SHOW FULL TABLES command as follows:

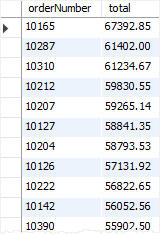
SHOW FULL TABLES;



The table\_type column in the result set specifies the type of the object: view or table (base table).

If you want to query total sales for each sales order, you just need to execute a simple SELECT  statement against the SalePerOrder  view as follows:

SELECT \* FROM salePerOrder;



### **2) Creating a view based on another view example**

MySQL allows you to create a view based on another view.

For example, you can create a view called bigSalesOrder based on the salesPerOrder view to show every sales order whose total is greater than 60,000 as follows:

CREATE VIEW bigSalesOrder AS

SELECT

orderNumber,

ROUND(total,2) as total

FROM

salePerOrder

WHERE

total > 60000;

Now, you can query the data from the bigSalesOrder view as follows:

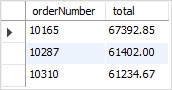
SELECT

orderNumber,

total

FROM

bigSalesOrder;



### **3) Creating a view with join example**

The following example uses the CREATE VIEW statement to create a view based on multiple tables. It uses the INNER JOIN clauses to join tables.

CREATE OR REPLACE VIEW customerOrders AS

SELECT

orderNumber,

customerName,

SUM(quantityOrdered \* priceEach) total

FROM

orderDetails

INNER JOIN orders o USING (orderNumber)

INNER JOIN customers USING (customerNumber)

GROUP BY orderNumber;

This statement selects data from the customerOrders view:

SELECT \* FROM customerOrders

ORDER BY total DESC;

This picture shows the partial output:



### **4) Creating a view with a subquery example**

The following example uses the CREATE VIEW statement to create a view whose SELECT statement uses a subquery. The view contains products whose buy prices are higher than the average price of all products.

CREATE VIEW aboveAvgProducts AS

SELECT

productCode,

productName,

buyPrice

FROM

products

WHERE

buyPrice > (

SELECT

AVG(buyPrice)

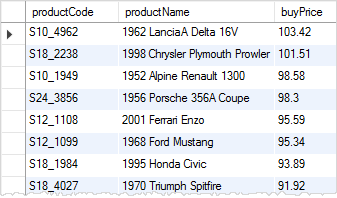
FROM

products)

ORDER BY buyPrice DESC;

This query data from the aboveAvgProducts is simple as follows:

SELECT \* FROM aboveAvgProducts;



### **5) Creating a view with explicit view columns example**

This statement uses the CREATE VIEW statement to create a new view based on the customers and orders tables with explicit view columns:

CREATE VIEW customerOrderStats (

customerName ,

orderCount

)

AS

SELECT

customerName,

COUNT(orderNumber)

FROM

customers

INNER JOIN

orders USING (customerNumber)

GROUP BY customerName;

This query returns data from the customerOrderStats view:

SELECT

customerName,

orderCount

FROM

customerOrderStats

ORDER BY

orderCount,

customerName;

